# Agentenbasierte Modellierung und Simulation schwarmintelligenter Algorithmen am Beispiel der Futtersuche von Ameisen und Bienen

## Studienarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Tim Saupp**

Abgabedatum 18.05.2018

| | |
|---|---|
| Bearbeitungszeitraum | 30.10.2017-18.05.2018 |
| Matrikelnummer | 2742603 |
| Kurs | TINF15B3 |
| Gutachter der Studienakademie | Prof. Dr. Lausen |

## Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Titel: Agentenbasierte Modellierung und Simulation schwarmintelligenter Algorithmen am Beispiel der Futtersuche von Ameisen und Bienen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum

Unterschrift

**Zusammenfassung**

In dieser Studienarbeit wird die Futtersuche von Ameisen und Bienen, anhand eines agentenbasierten Modells, simuliert. Dabei werden die Strategien der Tierschwärme für die Nahrungssuche berücksichtigt. Zur Implementierung der Modelle wird die MASON Java-Bibliothek, die ein breitgefächertes Repertoire an Funktionen für die ABM bereitstellt, verwendet.

# Inhaltsverzeichnis


**1 Projektbeschreibung** **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel der Arbeit . . . 1
- 1.3 Kapitelübersicht . . . 1

**2 Grundlagen** **2**
- 2.1 Schwarmintelligente Superorganismen . . . 2
  - 2.1.1 Begriffsdefintion . . . 2
  - 2.1.2 Ameisen . . . 3
  - 2.1.3 Bienen . . . 3
- 2.2 Agentenbasierte Modellierung . . . 4
  - 2.2.1 Agenten . . . 4
  - 2.2.2 Agenten-Beziehungen . . . 4
  - 2.2.3 Agenten-Umgebung . . . 4

**3 Schwarmintelligente Algorithmen** **5**
- 3.1 Particle Swarm Optimization . . . 5
  - 3.1.1 Suchraum . . . 5
  - 3.1.2 Partikel . . . 5
  - 3.1.3 Algorithmus . . . 6
- 3.2 Ant Colony Optimization . . . 7
  - 3.2.1 Suchraum . . . 7
  - 3.2.2 Modellierte Ameise . . . 7
  - 3.2.3 Algorithmus . . . 8
- 3.3 Bee Colony Optimization . . . 9
  - 3.3.1 Suchraum . . . 9
  - 3.3.2 Modellierte Biene . . . 9
  - 3.3.3 Algorithmus . . . 10

**4 Modelle und Frameworks** **11**
- 4.1 Modelle . . . 11
  - 4.1.1 Finden des globalen Minimums einer Funktion mit der PSO . . . 11
  - 4.1.2 Die Futtersuche von Ameisen und Bienen . . . 12
- 4.2 Framework . . . 13
  - 4.2.1 Anforderungen . . . 13
  - 4.2.2 Übersicht . . . 14
  - 4.2.3 Auswahl . . . 15

**5 Implementierung** **16**
- 5.1 Finden des globalen Minimums der Rastrigin-Funktion mit der PSO . . . 16
  - 5.1.1 Initalisiere Partikelschwarm im Suchraum . . . 16
  - 5.1.2 Bestimme Position und Geschwindigkeit . . . 16
  - 5.1.3 Bestimme den Fitnesswert . . . 17
  - 5.1.4 Bestimme beste Position . . . 17
  - 5.1.5 Berechne neue Geschwindigkeit . . . 17
  - 5.1.6 Berechne neue Position . . . 18
- 5.2 Die Futtersuche von Ameisen . . . 18
  - 5.2.1 Kehre zum Ameisenbau zurück . . . 18
  - 5.2.2 Hinterlasse Futterpheromone . . . 19
  - 5.2.3 Finde Futterquelle . . . 19

5.2.4 Hinterlasse Heimatpheromone . . . 20
5.2.5 Parameter $\beta$ . . . 20
5.2.6 Parameter $\alpha$ . . . 20
5.2.7 Verdunstung der Pheromone . . . 21
5.3 Die Futtersuche von Bienen . . . 21
5.3.1 Zufällige Suche nach der Futterquelle . . . 21
5.3.2 Bewegung in Richtung der Futterquelle . . . 22
5.3.3 Finden der Futterquelle . . . 22
5.3.4 Rückkehr zum Bienenstock . . . 22
5.3.5 Der Tanz . . . 23
5.3.6 Beobachtung des Tanzes . . . 23

**6 Experimente** **24**
6.1 Finden des globalen Minimums der Rastrigin-Funktion mit PSO . . . 24
6.1.1 Auswirkungen der Parameter auf die Ergebnisse der Simulation . . . 24
6.2 Futtersuche von Ameisen . . . 24
6.2.1 Auswirkungen der Parameter auf die Ergebnisse der Simulation . . . 24
6.3 Futtersuche von Bienen . . . 25

**7 Zusammenfassung** **26**

# Abbildungsverzeichnis


1 Entstehungsbedingung und Definition kollektiver Intelligenz bei Tierschwärmen 2
2 Der Bienentanz . . . . . 3
3 PSO, Berechnung von $x_i(t+1)$ und $v_i(t+1)$ . . . . . 6
4 zweidimensionale Rastrigin-Funktion . . . . . 11
5 Bildung einer Ameisenstraße in der Simulation der Futtersuche von Ameisen . . 24
6 Einfluss des Parameters $\alpha$ in der Simulation der Futtersuche von Ameisen . . . 25
7 Abbau der Futterquelle in der Simulation der Futtersuche von Bienen . . . . . 25

# Tabellenverzeichnis


1 Auswahl des Frameworks . . . . . 15
2 Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7 . . . . . 30
3 Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7 . . . . . 30
4 Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5 . . . . . 31
5 Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5 . . . . . 31
6 Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3 . . . . . 32
7 Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3 . . . . . 32
8 Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.3, $\beta$ = 0.7 . . . . . 33
9 Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.3, $\beta$ = 0.7 . . . . . 33
10 Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.5, $\beta$ = 0.5 . . . . . 34
11 Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.5, $\beta$ = 0.5 . . . . . 34
12 Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.7, $\beta$ = 0.3 . . . . . 35
13 Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.7, $\beta$ = 0.3 . . . . . 35
14 Ergebnisse Simulation der Futtersuche von Bienen: 2000 Bienen, 5000 Iterationen 36
15 Ergebnisse Simulation der Futtersuche von Bienen: 1000 Bienen, 5000 Iterationen 36
16 Ergebnisse Simulation der Futtersuche von Bienen: 100 Bienen, 5000 Iterationen 36

# Listings


1 PSO: Initalisierte Partikelschwarm im Suchraum . . . . . 16
2 PSO: Bestimme Position und Geschwindigkeit . . . . . 16
3 PSO: Bestimme Fitnesswert . . . . . 17
4 PSO: Bestimme beste Position . . . . . 17
5 PSO: Berechne neue Geschwindigkeit . . . . . 17
6 PSO: Berechne neue Position . . . . . 18
7 Ameisen: Kehre zum Ameisenbau zurück . . . . . 18
8 Ameisen: Hinterlasse Futterpheromone . . . . . 19
9 Ameisen: Finde Futterquelle . . . . . 19
10 Ameisen: Hinterlasse Heimatpheromone . . . . . 20
11 Ameisen: Parameter beta . . . . . 20
12 Ameisen: Parameter alpha . . . . . 20
13 Ameisen: Verdunstung der Pheromone . . . . . 21
14 Bienen: Zufällige Suche nach der Futterquelle . . . . . 21
15 Bienen: Bewegung in Richtung der Futterquelle . . . . . 22
16 Bienen: Finden der Futterquelle . . . . . 22
17 Bienen:Rückkehr zum Bienenstock . . . . . 22
18 Bienen: Der Tanz . . . . . 23
19 Bienen: Beobachtung des Tanzes . . . . . 23

# Abkürzungsverzeichnis

**ABM** Agentebasierte Modellierung

**ACO** Ameisenkolonieoptimierung

**BCO** Bienenkolonieoptimierung

**MASON** Multi-Agent Simulator Of Networks

**PSO** Partikelschwarmoptimierung

# 1 Projektbeschreibung

## 1.1 Motivation

In vielen Bereichen der Technik dient die Natur als Vorbild. Ameisen- und Bienenkolonien besitzen komplex wirkende Strategien, um überlebensnotwendige Aufgaben auf eine intelligente und möglichst effiziente Weise zu bewältigen. Dabei besitzt das Einzeltier nur ein sehr beschränktes Repertoire an Verhaltensregeln. Erst das selbstorganisierte Zusammenspiel vieler Einzeltiere führt zu den in der Natur beobachtbaren, intelligent wirkenden Strategien. Das Herausbilden von neuen Eigenschaften und Strukturen eines Systems infolge des Zusammenspiels seiner Elemente wird als Emergenz bezeichnet.

Für die Informatik ist die Emergenz für ein breites Spektrum an Anwendungsfällen interessant. Die Übertragung der Idee, komplexe Systeme auf einfache, regelbasierte Komponenten aufzuteilen könnte beispielsweise dazu genutzt werden, der stets ansteigenden Systemkomplexität entgegenzuwirken.

## 1.2 Ziel der Arbeit

Ziel der Studienarbeit ist die Modellierung und Simulation der Futtersuche von Ameisen und Bienen anhand eines agentenbasierten Modells. Dafür sollen die Strategien der Tiere, zur Entwicklung einer möglichst effizienten Methode für den Abbau einer Futterquelle, angewandt werden. Als abstraktes Beispiel wird zudem ein Modell der PSO implementiert, um das globale Minimum der Rastrigin-Funktion zu finden. Im Anschluss an die Implementierung der Modelle werden Experimente durchgeführt.

## 1.3 Kapitelübersicht

Kapitel 2 erläutert den Begriff schwarmintelligente Superorganismen, definiert den Begriff Schwarmintelligenz und beschreibt die Grundlagen der agentenbasierte Modellierung. In Kapitel 3 werden die schwarmintelligenten Algorithmen PSO, ACO und BCO vorgestellt. Die implementierten Modelle werden in Kapitel 4 definiert. Zudem wird die Auswahl des verwendeten Frameworks, zur Implementierung der definierten Modelle, gerechtfertigt. Nachfolgend wird in Kapitel 5 die Implementierung der Modelle beschrieben. Kapitel 6 dokumentiert die durchgeführten Simulationen der implementierten Modelle. In Kapitel 7 folgt die Zusammenfassung der Arbeit.

# 2 Grundlagen

## 2.1 Schwarmintelligente Superorganismen

Durch die sensuale Verbindung der Tiere wird die Futtersuche und das Abwehren von Gefahren ohne eine zentrale Lenkung bzw. ohne hierarchische Befehlskette bewältigt. Instinktiv verankerte Regeln sorgen dafür, dass auf bestimmte Aktionen der Tiere in vollkommen deterministischer Weise eine Reaktion erfolgt. Aus dieser dezentralen Interaktion entstehen, bei einem Kollektiv von Tieren, intelligente Resultate auf Makroebene[1].

### 2.1.1 Begriffsdefintion

Bereits 1911 bezeichnet W. M. Wheeler, amerikanischer Ethologe mit Spezialisierung auf dem Gebiet der Erforschung sozialer Insekten, Kolonien wie die der Bienen und Ameisen als Superorganismen mit emergenten Fähigkeiten[2]. Abbildung 1 zeigt die Definition und Entstehungsbedingung kollektiver Intelligenz bei Tierschwärmen:

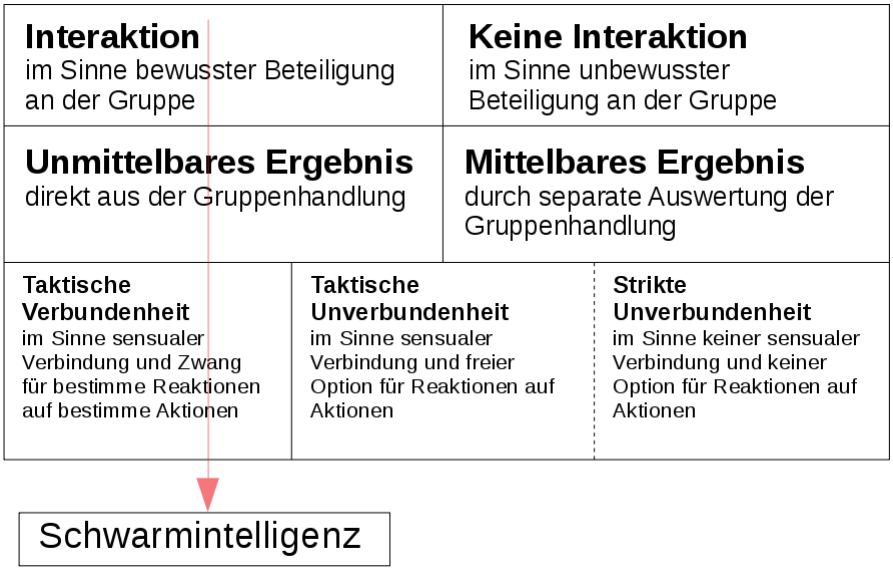


Abbildung 1: Entstehungsbedingung und Definition kollektiver Intelligenz bei Tierschwärmen[3]

- **Interaktion:** Auf Aktion und Reaktion basierende Interaktion gilt als Grundlage für die Definition des Begriffs Schwarmintelligenz. Anlass dafür ist der in den Tieren vorhandene Überlebensinstinkt, der diese veranlasst, sich bewusst an der Gruppe zu beteiligen.
- **Unmittelbares Ergebnis:** Einzelne Tiere führen Handlungen aus ohne Wissen um das Schwarmergebnis. Das Resultat entsteht unmittelbar aus der Handlung des Schwarms und bedarf keiner externen Aggregation und Auswertung.
- **Taktische Verbundenheit:** Sowohl die sensuale Verbindung der Tiere, als auch, der in den Tieren verankerte Instinkt zeugen von der taktischen Verbundenheit des Schwarms bestehend aus festen Aktions- und Reaktionsmustern.

[1] vgl. [1], A. Aulinger (2013)
[2] [11], W.M. Wheeler (1911)
[3] [1], A. Aulinger (2013)

Zusammenfassend beschreibt der Begriff Schwarmintelligenz ein Phänomen aus dem Tierreich zur Selbstorganisation eines Schwarms. Damit lebensnotwendige Aufgaben gemeinsam und auf intelligente Weise bewältigen werden. Dabei vollbringen die Tiere im Schwarm Leistungen, die das Vermögen jedes Einzeltiers übersteigen.

### 2.1.2 Ameisen

Ameisen bilden Staaten mit einigen hundert bis zu mehreren Millionen Individuen. Trotz dieser riesigen Anzahl funktioniert ein Ameisenstaat, da er sich selbst organisiert ohne eine hierarchische Instanz, die einen Überblick über alle Aufgaben besitzt oder diese steuert und verteilt. Stattdessen führen die Handlungen einzelner Ameisen im Zusammenspiel zu einem organisierten Staat, der für die Ameisen sorgt und Nahrung, Brutpflege und Schutz bietet. Besonders interessant ist die Fähigkeit effiziente Wege zwischen Futterquelle und Ameisenbau zu finden.[4].

Ist der Futtervorrat des Ameisenbaus erschöpft verlassen mehrere Ameisen den Ameisenbau gleichzeitig und begeben sich auf die Futtersuche. Sobald eine Ameise eine Futterquelle gefunden hat, nimmt sie eine Gewichtseinheit des Futters mit und begibt sich auf den Rückweg zum Ameisenbau. Dabei setzt die Ameise Pheromone frei die mit der Zeit verfliegen, um den Weg zur Futterquelle zu markieren. Die Ameise die den kürzesten Weg zu einer Futterquelle gefunden hat legt die Strecke zwischen Ameisenbau und Futterquelle häufiger zurück. Die Pheromonspur wird durch das häufige Zurücklegen der Strecke intensiviert und dient als sicherer Wegweiser zur Futterquelle. Mitglieder der Kolonie folgen den intensivsten Pheromonspuren. Ist die Futterquelle erschöpft, löst sich die Pheromonspur auf[5].

### 2.1.3 Bienen

Im kilometerweiten Gelände besitzt keine Biene den gesamten geographischen Überblick. Sie entscheidet nur lokal über die Rentabilität der Futterquelle. Kundschafterinnen und Sammlerinnen, die von der Futtersuche wiederkehren, führen den in Abbildung 2 dargestellten Tanz auf und zeigen unbeschäftigten Bienen damit die Richtung und Entfernung zur gefundenen Futterquelle. Unbeschäftigte Bienen sehen sich die Tänze der im Bienenstock eintreffenden Bienen an und entscheiden sich anschließend für ihr nächstes Ziel. Sobald eine Futterquelle erschöpft ist brauchen die Sammlerinnen länger beim Sammeln und veranlassen aufgrund der geringeren Anzahl an Tänzen weniger Bienen dazu am gleichen Ort zu sammeln[6].

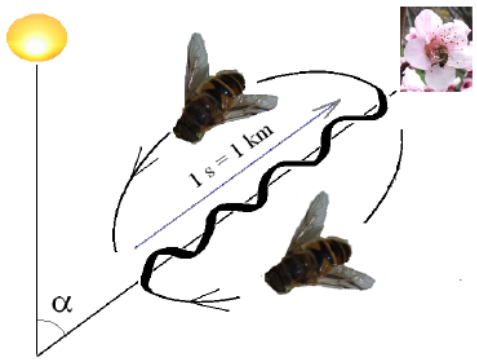


Abbildung 2: Der Bienentanz[7]

[4]vgl. [7], L. Pintscher (2008)
[5]vgl. [11], R. Wehner (2001)
[6]vgl. [7], L. Pintscher (2008)
[7][7], L. Pintscher (2008)

## 2.2 Agentenbasierte Modellierung

Die Agentebasierte Modellierung (ABM) erlaubt eine natürliche Beschreibung von Systemen als eine Sammlung autonomer entscheidungsfähiger Agenten in einer gemeinsamen Umgebung, um emergente Phänomene zu analysieren[8]. Das Ziel der ABM besteht darin, durch die Simulation einer Vielzahl an Agenten, das resultierende Systemverhalten zu untersuchen. Nach C. Macal und M. North[9] verfügt ein agentenbasiertes Modell über folgende Komponenten:

- **Agenten:** Agenten handeln autonom, proaktiv und reaktionär auf der Basis von festgelegten Regeln. Jeder Agent besitzt nur eine eingeschränkte Sicht auf das Gesamtsystem. Sein Wissen über den globalen Zustand des Systems ist immer unvollständig.
- **Agenten-Beziehungen:** Beziehungen zwischen Agenten entstehen durch die proaktive Aktion eines Agenten und die darauffolgende Reaktion eines anderen Agenten oder Interaktionen der Agenten mit der Umgebung.
- **Agenten-Umgebung:** Agenten interagieren innerhalb einer Umgebung mit anderen Agenten und ihrer Umgebung.

### 2.2.1 Agenten

M. Wooldridge und N. Jennings[10] weisen Agenten die Charaktereigenschaften Autonomie, Proaktivität, Reaktivität und die Fähigkeit zur Interaktion durch Kommunikation zu. Die Handlungsautonomie eines Agenten beschränkt sich auf die Fähigkeit, eigenständig zu entscheiden, welche der ihm zur Verfügung stehenden Aktionen situationsbedingt auszuführen ist. Zielvorgaben bestimmen wann ein Agent welche Aktionen ausführt. Proaktivität beschreibt dabei das zielgerichtete Verhalten eines Agenten der selbst aktiv wird, statt nur auf die Umgebung zu reagieren.

### 2.2.2 Agenten-Beziehungen

Zwischen den Systemelementen bestehen Interaktionsbeziehungen. Die technischen Voraussetzungen für die Kommunikation werden von der Umgebung bereitgestellt. Beziehungen lassen sich einteilen in Interaktion zwischen Agenten und Umgebung, Interaktion zwischen Agenten und organisatorisch bedingte Beziehungen. Die Kommunikation zwischen Agenten kann dabei direkt durch Austausch von Nachrichten oder indirekt über die Veränderung der Umgebung erfolgen.

### 2.2.3 Agenten-Umgebung

Nach S. Russell und P. Norvig[11] muss eine geeignete Agenten-Umgebung zugänglich, deterministisch, dynamisch, kontrollierbar und teleologisch sein. Eine zugängliche Umgebung erlaubt den Agenten Zugriff auf ihren Zustand. In der Regel beschränkt sich dieser Zugriff jedoch auf den lokalen Wahrnehmungsbereich eines Agenten und erlaubt somit nur Zugriff auf einen Ausschnitt der Umgebung. Deterministisch ist die Umgebung, sobald der Folgezustand vollständig durch den aktuellen Umgebungszustand und die aktuelle Aktion des Agenten bestimmt ist. Kann sich der Zustand der Umgebung durch Aktionen der Agenten ändern, so handelt es sich um eine dynamische und kontrollierbare Umgebung.

[8] vgl. [2], E. Bonabeau (2002)
[9] [6], C. Macal, M. North (2010)
[10] [14], M. Wooldridge, N. Jennings (1995)
[11] [9], S. Russel, P. Norvig (2009)

# 3 Schwarmintelligente Algorithmen

Die schwarmintelligenten Algorithmen Partikelschwarmoptimierung (PSO), Ameisenkolonieoptimierung (ACO) und Bienenkolonieoptimierung (BCO) orientieren sich an den Strategien der Tierschwärme in der Natur. Die biologische Plausibilität spielt dabei eine untergeordnete Rolle. Im Vordergrund steht die Übertragung der Strategien der Tierschwärme, um Approximationsalgorithmen für Optimierungsprobleme zu entwerfen. Die PSO wird in Rahmen dieser Studienarbeit implementiert. Die ACO und BCO werden vorgestellt um das Verhalten der Tiere und die Übertragung des Verhaltens in Algorithmen besser verstehen zu können.

## 3.1 Particle Swarm Optimization

Die PSO wurde erstmals im Jahr 1995 von J. Kennedy und R. Ebert[12] beschrieben. Sie stellt ein, auf der Abfolge von abstrakten Schritten basiertes, Verfahren zur näherungsweisen Lösung von Optimierungsproblemen dar. Zur Lösung des Optimierungsproblems wird eine Population von Partikeln, so lange durch einen Suchraum bewegt, bis eine hinreichende Lösung in einer endlichen Zeit gefunden wird. Dabei stellt die Position eines Partikels eine potentielle Lösung des Optimierungsproblems dar und wird in jedem Zeitschritt neu berechnet[13].

### 3.1.1 Suchraum

Der Suchraum $S \subset \mathbb{R}^n$ ist ein n-dimensionaler Raum, in dem sich die gesamte Population $X = \{x_1, x_2, ..., x_n\}$ der Partikel bewegt. Damit eine Lösung gefunden werden kann beschränkt sich der Suchraum auf eine endliche Anzahl an möglichen Positionen, die von den Partikeln besucht werden können.

### 3.1.2 Partikel

Jeder Partikel $x_i$ besitzt folgende Eigenschaften im Suchraum $S$ zum Zeitpunkt $t$:

- **Position $x_i(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position des Partikels $x_i$ zum Zeitpunkt $t$ repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_{b,i}(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_{b,i}(t))$:** Die Position $x_{b,i}$ ist die bisher beste Position des Partikels $x_i$ zum Zeitpunkt $t$.
- **Position $x_g(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_g(t))$:** Die Position $x_g$ ist die bisher beste Position des Partikelschwarms zum Zeitpunkt $t$.
- **Geschwindigkeit $v_i(t)$:** Die Geschwindkeit $v_i$ mit der sich der Partikel durch den Suchraum $S$ zum Zeitpunkt $t$ bewegt.
- **Trägheitskoeffizient $w$:** Der Trägheitskoeffizient $w$ bestimmt die Gewichtung der Geschwindigkeit $v_i(t)$ zum Zeitpunkt $t + 1$.
- **lokaler Vertrauenskoeffizient $a_l$:** Der lokale Vertrauenskoeffizient $a_l$ bestimmt die Gewichtung der lokalen Attraktion, bei der Berechnung der Geschwindigkeit $v_i(t + 1)$.
- **globaler Vertrauenskoeffizient $a_g$:** Der globale Vertrauenskoeffizient $a_g$ bestimmt die Gewichtung der globalen Attraktion, bei der Berechnung der Geschwindigkeit $v_i(t + 1)$.

Der Trägheitskoeffizient $w$ und die Vertrauenskoeffizienten $a_l$ und $a_g$ geben an, wie sehr die Partikel sich selbst, ihren eigenen Erfahrungen und den Erfahrungen ihrer Nachbarn vertrauen.

[12] [4], J. Kennedy, R. Ebert (1995)
[13] vgl. [5], Y. Liu (2014)

### 3.1.3 Algorithmus

Initialisiere Partikelschwarm $X$ im Suchraum $S$;
**while** *Maximale Iterationen nicht erreicht* **do**
  Bestimme für jeden Partikel Position $x_i(t)$;
  Bestimme für jeden Partikel Geschwindigkeit $v_i(t)$;
  Berechne für jeden Partikel den Funktionswert $f(x_i(t))$;
  **if** $f(x_i(t))$ *besser als* $f(x_{b,i}(t))$ **then** Setze $x_{b,i}(t) = x_i(t)$ ;
  **if** $f(x_i(t))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t)) = x_i(t)$ ;
  Berechne neue Geschwindigkeit $v_i(t+1)$ des Partikels $x_i$;
  Berechne neue Position $x_i(t+1)$ des Partikels $x_i$;
**end**

**Algorithm 1:** PSO Algorithmus

Der abgebildete Algorithmus 1 zeigt den Ablauf der PSO. Im ersten Schritt wird die Population $X$ mit zufälligen Positionen $x_i$ im Suchraum $S$ initialisiert. Bis die Anzahl an maximalen Iterationen nicht erreicht ist wird für jeden Partikel geprüft ob der Funktionswert $f(x_i(t))$ für die Position $x_i(t)$ besser ist als der Funktionswert $f(x_i(t-1))$ im vorherigen Schritt an der Position $x_i(t-1)$. Ist der Funktionswert $f(x_i(t))$ für die Position $x - i(t)$ des Partikels besser, wird die bisher beste Position des Partikels $x_{b,i}(t)$ zu $x_i(t)$. Zudem wird für jeden Partikel $x_i$ geprüft, ob der aktuelle Funktionswert $f(x_i(t))$ des Partikels besser ist als der globale Funktionswert $f(x_g(t))$. Ist dies der Fall so wird die global beste Position $x_g(t)$ zu $x_i(t)$. Für die nächste Iteration wird die Geschwindigkeit $v_i(t+1)$ und die neue Position $x_i(t+1)$ des Partikels nach folgenden Formeln berechnet:

$$v_i(t+1) = wv_i(t) + a_l r_1(x_{b,i}(t) - x_i(t)) + a_g r_2(x_g(t) - x_i(t)) \tag{1}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2}$$

Bei der Berechnung der neuen Geschwindigkeit $v_i(t+1)$ wird die Entfernung der Position $x_g(t)$ zur Position $x_{b,i}(t)$ berechnet. Dabei fallen die Zufallszahlen $r_1, r_2 \subset \mathbb{R}$ ins Gewicht, sowie die Parameter $w$, $a_l$, $a_g$, mit denen das Verhalten des Schwarms verändert wird. Gilt $a_l > a_g$, dann streben die Partikel in Richtung von $x_{b,i}(t)$. Ist $a_g > a_l$, dann konvergieren die Partikel zur Position $x_g(t)$. Abbildung 2 zeigt in grafischer Darstellung die Berechnung der neuen Geschwindigkeit $v_i(t+1)$ und der neuen Position $x_i(t+1)$.

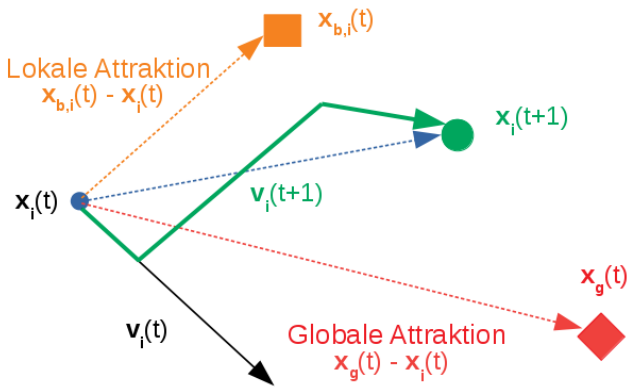


Abbildung 3: Berechnung von $x_i(t+1)$ und $v_i(t+1)$

## 3.2 Ant Colony Optimization

Die von M. Dorigo vorgestellte ACO dient, wie die PSO, zur Lösung von Optimierungsproblemen. Zur näherungsweisen Lösung des Optimierungsproblems wird bei der ACO eine Population an modellierten Ameisen instanziiert und durch den, vom Optimierungsproblem definierten, Suchraum bewegt. Die Positionen der modellierten Ameisen repräsentieren potentielle Lösungen des Optimierungsproblems und werden in jedem Zeitschritt neu berechnet. Zudem ist jede Position im Suchraum mit einem Pheromonniveau markiert[14].

### 3.2.1 Suchraum

Der Suchraum $S \subset \mathbb{R}^n$ ist ein n-dimensionaler Raum, in dem sich die gesamte Population $X = \{x_1, x_2, ..., x_n\}$ der modellierten Ameisen bewegt. Damit eine Lösung gefunden werden kann beschränkt sich der Suchraum auf eine endliche Anzahl an möglichen Positionen $P = \{p_1, p_2, ...., p_n\}$ die von den modellierten Ameisen besucht werden können. Jeder Position $P$ im Suchraum $S$ ist ein Wert $\tau_i$ zugeordnet, der das Pheromonniveau der jeweiligen Position beschreibt.

### 3.2.2 Modellierte Ameise

Jede modellierte Ameise besitzt folgende Eigenschaften im Suchraum $S$ zum Zeitpunkt $t$:

- **Position $x_i(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position der modellierten Ameise $x_i$ zum Zeitpunkt $t$ repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_g(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_g(t))$:** Die Position $x_g$ ist die bisher beste Position der Ameisenkolonie zum Zeitpunkt $t$.
- **Teillösungen $C = \{c_1(t), c_2(t), ..., c_n(t)\}$ die der modellierten Ameise $x_i$ zum Zeitpunkt $t$ zur Verfügung stehen:** Die modellierte Ameise $x_i$ wählt aus der ihr zur Verfügung stehenden Teillösungen $C$ zum Zeitpunkt $t$ eine Teillösung. Die gewählte Teillösung entspricht der nächsten Position der modellierten Ameise $x_i$ zum Zeitpunkt $t + 1$.
- **Wahrscheinlichkeiten $W = \{w_{c_1}(t), w_{c_2}(t), ..., w_{c_n}(t)\}$ zur Wahl von $x_i(t + 1)$:** Die Wahrscheinlichkeiten $W$ werden für die, der modellierten Ameise zur Verfügung stehenden, Teillösungen $C$ berechnet. Die Teillösung mit der höchsten Wahrscheinlichkeit wird von der modellierten Ameise ausgewählt.
- **Parameter $\alpha$:** Der Parameter $\alpha$ bestimmt die Gewichtung der Pheromonmarkierungen der Teillösungen $C$ bei der Berechnung der Wahrscheinlichkeiten $W$.
- **Parameter $\beta$:** Der Parameter $\beta$ bestimmt die Gewichtung der Güte der Teillösungen $C$ bei der Berechnung der Wahrscheinlichkeiten $W$

Mit $\alpha = 0$ und $\beta = 1$ erhält man einen Greedy-Algorithmus. Umgekehrt erhält man mit $\alpha = 1$ und $\beta = 0$ einen rein zufallsgesteuerten Algorithmus.

[14] vgl. [3], M. Dorigo, M. Birattari, T. Stützle (2004)

### 3.2.3 Algorithmus

Initialisiere Ameisenkolonie $X$ im Suchraum $S$;
**while** *Maximale Iterationen nicht erreicht* **do**
  Berechne für jede modellierte Ameise $x_i$ die Wahrscheinlichkeiten $W$ für die möglichen Teillösungen $C$ ausgehend von Position $x_i(t)$;
  Bestimme für jede modellierte Ameise neue Position $x_i(t+1)$;
  **if** $f(x_i(t+1))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t))$ = $x_i(t+1)$ ;
  Berechne für jede gewählte Teillösung die neue Pheromonmarkierung $\tau_i(t+1)$ für Position $P$ im Suchraum $S$;
**end**

**Algorithm 2:** ACO Algorithmus

Im ersten Schritt des abgebildeten Algorithmus 2 wird die Population $X$ an einer bestimmten Position $P$ im Suchraum $S$ initialisiert. Solange die Anzahl an maximalen Iterationen nicht erreicht ist berechnet jede Ameise die Wahrscheinlichkeiten $W$ für die ihr zur Verfügung stehenden Teillösungen $C$ mit der Formel:

$$W(c_i|x_i(t)) = \frac{\tau_{c_i}^{\alpha}(t) \cdot [f(c_i(t))]^{\beta}}{\sum c \in C\ \tau_c(t)^{\alpha} \cdot [f(c(t))]^{\beta}} \tag{1}$$

Die Wahrscheinlichkeit ist abhängig von dem Pheromonniveau $tau_{c_i}$ der jeweiligen Teillösung und der Güte der Teillösung, bestimmt durch $f(c_i(t))$. Die Parameter $\alpha$ und $\beta$ verändern das Verhalten der modellierten Ameisen. Gilt $\alpha > \beta$ dann ist das Pheromonniveau der Teillösung ausschlaggebender als die Güte der Teillösung für die Berechnung der Wahrscheinlichkeiten $W$. Sind die Wahrscheinlichkeiten $W$ für alle Teillösungen $C$ berechnet, entscheidet sich die modellierte Ameise für die Teillösung mit der höchsten Wahrscheinlichkeit. Die von der modellierten Ameise gewählte Teillösung entspricht der neuen Position $x_i(t+1)$. Im Anschluss wird für jede modellierte Ameise $x_i(t)$ geprüft, ob der aktuelle Funktionswert $f(x_i(t))$ besser ist als der globale Funktionswert $f(x_g(t))$. Ist dies der Fall so wird die global beste Position $x_g(t)$ zu $x_i(t)$. Am Ende der Iteration werden die Pheromonmarkierungen mit folgender Formel aktualisiert:

$$\tau_i(t+1) = (1-\rho)\tau_i(t) + \sum_{x_i \in X} \Delta\tau_i(t)^C \tag{2}$$

Hierbei ist $\rho$ der konstante Verdunstungsfaktor der Pheromonmarkierungen. Die an der gewählten Teillösung hinterlassene Pheromonmenge $\Delta\tau_i(t)^C$ berechnet sich durch die Formel:

$$\Delta\tau_i(t)^C = f(c_i(t)) \tag{3}$$

## 3.3 Bee Colony Optimization

Die im Zeitraum 1999 bis 2003 von D. Teodorović und P. Lučić[15] entwickelte BCO stellt, wie die PSO und ACO, ein metaheuristisches Verfahren zur Lösung von Optimierungsproblemen dar. Die zur näherungsweisen Lösung des Optimierungsproblems, durch den Suchraum bewegten modellierten Bienen repräsentieren potentielle Lösungen des Optimierungsproblems, über ihre Position im Suchraum.

### 3.3.1 Suchraum

Der Suchraum $S$ der BCO unterscheidet sich nicht von dem in Kapitel 3.1.1 beschriebenen Suchraum der PSO.

### 3.3.2 Modellierte Biene

Jede modellierte Biene besitzt folgende Eigenschaften im Suchraum $S$ zum Zeitpunkt $t$:

- **Position $x_i(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position der modellierten Biene $x_i$ zum Zeitpunkt $t$ repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_g(t)$ im Suchraum $S$ mit einem Funktionswert $f(x_g(t))$:** Die Position $x_g$ ist die bisher beste Position der Bienenkolonie zum Zeitpunkt $t$.
- **Loyalitätswahrscheinlichkeit $l_i(t)$:** Die Loyalitätswahrscheinlichkeit $l_i(t)$ der modellierten Biene $x_i$ bestimmt den Status $Z$ zum Zeitpunkt $t$.
- **Status $Z$:** Ist die Loyalitätswahrscheinlichkeit $l_i(t)$ größer als eine Zufallszahl so wird die modellierte Biene $x_i$ zum Zeitpunkt $t$ in den Status Loyal versetzt. Die modellierte Biene erhält den Status Frei sobald die Loyalitätswahrscheinlichkeit kleiner ist als die Zufallszahl.
- **(optional) Rekrutierungsbienen $R = \{r_1(t), r_2(t), ..., r_n(t)\}$ die der modellierten Biene $x_i$ zum Zeitpunkt $t$ zur Verfügung stehen:** Die modellierte Biene $x_i$ wählt aus der ihr zur Verfügung stehenden Rekrutierungsbienen $R$ zum Zeitpunkt $t$ eine Rekrutierungsbiene.
- **(optional) Richtung $v_i(t)$:** Die Richtung $v_i(t)$ von der aktuellen Position $x_i(t)$ der modellierten Biene zur gewählten Rekrutierungsbiene $r_i(t)$.

Befindet sich eine modellierte Biene $x_i$ im Status Loyal besitzt sie zum Zeitpunkt $t$ keine Rekrutierungsbienen $R$ sondern ist ihrer Lösung $x_i(t)$ gegenüber loyal und wird diese nicht verändern. Ebenso besitzt sie daher keine Richtung $v_i(t)$.

[15] [10], D. Teodorović, P. Lučić (2001)

### 3.3.3 Algorithmus

Initialisiere Bienenkolonie $X$ im Suchraum $S$;
**while** *Maximale Iterationen nicht erreicht* **do**
  Bestimme für jede modellierte Biene Position $x_i(t)$ und berechne den Funktionswert $f(x_i(t))$;
  Bestimme $n$ Rekrutierungsbienen $R$ mit den besten Funktionswerten $f(x_i(t))$;
  Berechne für jede modellierte Biene die Loyalitätswahrscheinlichkeit $l_i(t)$;
  **if** $l_i(t)$ *größer als Zufallszahl* **then**
    Versetze Biene in den Status Loyal;
  **else**
    Versetze Biene in den Status Frei;
  **end**
  **if** *Biene im Zustand Frei* **then**
    Berechne Rekrutierungswahrscheinlichkeiten $W$ für Rekrutierungsbienen $R$ und wähle Rekrutierungsbiene $r_i(t)$;
    Bestimme Richtung $v_i(t)$ von Position $x_i(t)$ zu $r_i(t)$;
    Bewege dich in Richtung $v_i(t)$;
  **else**
    Setze neue Position $x_i(t+1) = x_i(t)$;
  **end**
  **if** $f(x_i(t+1))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t)) = x_i(t+1)$ ;
**end**

**Algorithm 3:** BCO Algorithmus

Der erste Schritt des abgebildeten Algorithmus 3 ist die Initialisierung der modellierten Bienen im Suchraum $S$. Bis die maximale Anzahl an Iterationen nicht erreicht ist wird in jeder Iteration die aktuelle Position der modellierten Biene und der dazugehörige Funktionswert $f(x_i(t))$ der potentiellen Lösung bestimmt. Im Anschluss werden $n$ modellierte Bienen mit den besten Funktionswerten ausgewählt. Diese Bienen stellen die Rekrutierungsbienen $R$ dar. Für jede modellierte Biene, ausgenommen die Rekrutierungsbienen, wird im nächsten Schritt die Loyalitätswahrscheinlichkeit $l_i(t)$ nach folgender Formel berechnet:

$$l_i(t) = e^{-\frac{f(x_g(t))-f(x_i(t))}{AnzahlderIterationen}} \tag{1}$$

Die modellierten Bienen werden nach der Berechnung der Loyalitätswahrscheinlichkeit $l_i(t)$ in den Status Loyal oder Frei versetzt. Befindet sich eine modellierte Biene im Status Loyal hält sie an ihrer Lösung $x_i(t)$ fest. Wird die modellierte Biene in den Status Frei versetzt werden die Rekrutierungswahrscheinlichkeiten zur Wahl der Rekrutierungsbienen nach folgender Formel berechnet:

$$r_i(t) = \frac{f(r_i(t)}{\sum_{r \in R}\ f(r(t))} \tag{2}$$

Jede modellierte Biene im Status Frei wählt anschließend eine Rekrutierungsbiene $r_i(t)$ auf der Grundlage der berechneten Rekrutierungswahrscheinlichkeiten. Im Anschluss wird die Richtung $v_i(t)$ von der Position modellierten Biene $x_i(t)$ zu der gewählten Rekrutierungsbiene $r_i(t)$ bestimmt. Im letzten Schritt der Iteration bewegt sich die modellierte Biene auf die Lösung ihrer Rekrutierungsbiene zu.

# 4 Modelle und Frameworks

## 4.1 Modelle

Als abstraktes Beispiel wird die PSO angeführt. Die PSO wird in dieser Arbeit dazu verwendet das globale Minimum der Rastrigin-Funktion zu finden. Bei der Modellierung der Futtersuche von Ameisen und Bienen sollen die Strategien der Tiere, zum effizienten Abbau einer Futterquelle, berücksichtigt werden. Das Modell der Futtersuche von Ameisen und Bienen soll keine Implementierung der vorgestellten schwarmintelligenten Algorithmen ACO und BCO darstellen, sondern eine möglichst realitätsnahe Simulation der Futtersuche der Tiere in der Natur zulassen.

### 4.1.1 Finden des globalen Minimums einer Funktion mit der PSO

**Modellbeschreibung** Im Modell werden Partikel durch die zweidimensionale Agenten - Umgebung bewegt, um das globale Minimum der, in Abbildung 3 abgebildeten, zweidimensionalen Rastrigin-Funktion zu finden. Die Rastrigin-Funktion wurde 1974 von A. Leonard vorgeschlagen und dient der Performanceanalyse von Optimierungsalgorithmen[16]. Die Rastrigin-Funktion ist definiert durch:

$$f(x) = An + \sum_{i=0}^{n} [x_i^2 - A\cos(2\pi x_i)] \tag{1}$$

Dabei ist $A = 10$ eine Konstante, $n$ die Dimensionen. Außerdem gilt $x = x_1, ..., x_n$ mit $x_i \in [-5.12, 5.12]$.

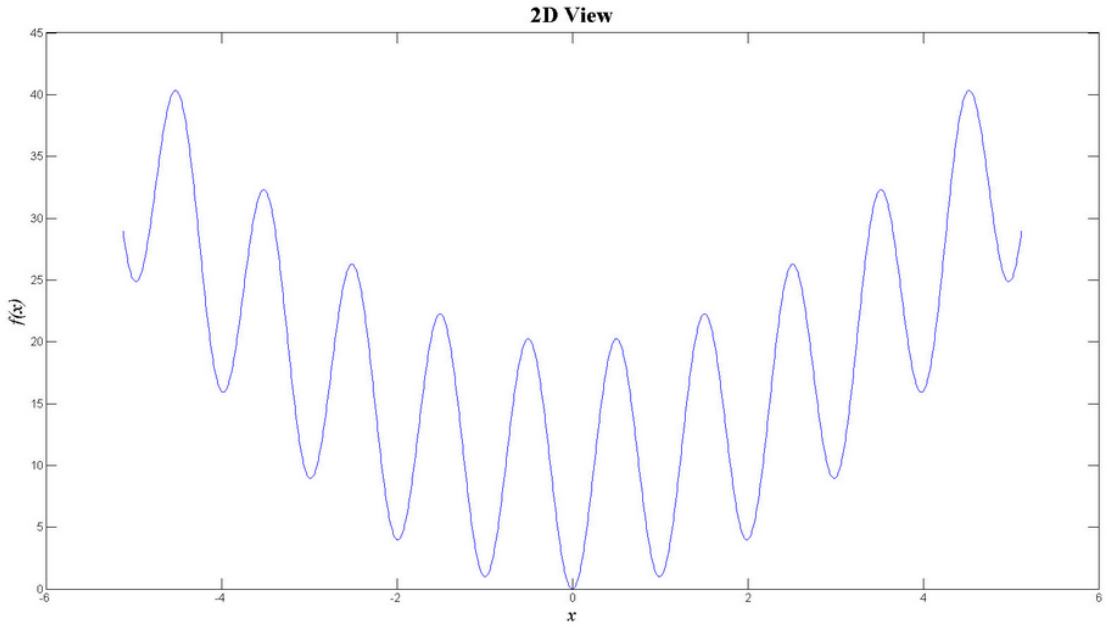


Abbildung 4: Zweidimensionale Rastrigin-Funktion[17]

**Methode** Das Modell zur Simulation der PSO ist in dieser Arbeit als abstraktes Beispiel für agentenbasierten Modellierung angeführt. Die Methode zum Finden des globalen Minimums der Rastrigin-Funktion stellt der im Kapitel 3.3 vorgestellte PSO-Algorithmus dar.

[16] [13], Wikipedia (2016)
[17] Bildquelle: http://al-roomi.org/benchmarks/unconstrained/n-dimensions/174-generalized-rastrigin-s-function

**Parameter** Im Modell ist die Anzahl der Partikel, der lokale Vertrauenskoeffizient und der globale Vertrauenskoeffizient einstellbar.

### 4.1.2 Die Futtersuche von Ameisen und Bienen

**Modellbeschreibung** Im Modell der Futtersuche suchen die Agenten nach einer Futterquelle in einer zweidimensionalen Agenten-Umgebung. Sie starten ihre Suche im Ameisenbau oder Bienenstock. Das erste Ziel des Modells ist das zufallsgesteuerte Finden der Futterquelle durch die Agenten. Im Anschluss interagieren die Agenten miteinander um die Futterquelle möglichst effizient abzubauen. Um die Effizienz der Methoden in Experimenten zu messen wird die Anzahl der gesammelten Futtereinheiten in einem bestimmten Zeitraum ermittelt.

**Methode der Ameisen** Zum effizienten Abbau der Futterquelle kommunizieren die modellierten Ameisen über die zweidimensionale Agenten-Umgebung in Form von Pheromonmarkierungen. Im Modell gibt es zwei Pheromontypen, die mit der Zeit verdunsten. Heimatpheromone markieren den Weg zum Ameisenbau und Futterpheromone den Weg zur Futterquelle. Auf der Suche nach der Futterquelle werden Heimatpheromone hinterlassen. Findet eine modellierte Ameise die Futterquelle wird eine Einheit der Futterquelle abgebaut. Nachdem die Futterquelle abgebaut wurde kehrt die Ameise zum Ameisenbau zurück. Dabei hinterlässt sie Futterpheromone. Befindet sich die modellierte Ameise im Ameisenbau liefert sie die Futtereinheit ab und begibt sich anschließend erneut auf Futtersuche. Der abgebildete Algorithmus 4 fasst das Vorgehen für den Abbau der Futterquelle zusammen.

**if** *hatFuttereinheit* **then**
    Kehre zum Ameisenbau zurück (Folge Heimatpheromonen);
    Hinterlasse Futterpheromone;
    **if** *Position der Ameise == Position des Ameisenbaus* **then** hatFuttereinheit = false ;
**else**
    Finde Futterquelle (Folge Futterpheromonen);
    Hinterlasse Heimatpheromone;
    **if** *Position der Ameise == Position der Futterquelle* **then** hatFuttereinheit = true ;
**end**

**Algorithm 4:** Futtersuche Ameisen

**Parameter Ameisen** Die einstellbaren Parameter des Modells sind die Anzahl der modellierten Ameisen und die Wahrscheinlichkeiten mit denen sich die Ameisen an die Pheromonmarkierungen folgen oder zufällig handeln in Form der Parameter $\alpha$ und $\beta$.

**Methode der Bienen** Um die Futterquelle möglichst effizient abzubauen kommunizieren die modellierten Bienen über den Bienentanz. Findet eine modellierte Biene die Futterquelle kehrt sie zum Bienenstock zurück. Dort tanzt sie für andere Bienen um ihnen die Richtung und Entfernung der Futterquelle mitzuteilen. Bienen die den Tanz beobachtet haben verlassen den Bienenstock und fliegen an die ihnen mitgeteilte Position. Um die exakte Position der Futterquelle ausfindig zu machen, suchen die modellierten Bienen im Umkreis der ihnen durch den Tanz mitgeteilten Position. Finden die modellierten Bienen die Futterquelle kehren sie zum Bienenstock zurück und tanzen, um weiteren Bienen die Position der Futterquelle mitzuteilen. Der abgebildete Algorithmus 5 fasst das Vorgehen der Bienen zusammen.

```
if InformationFutterquelle then
    Bewege dich in die Richtung der Futterquelle;
    Finde die Futterquelle;
    Kehre zum Bienenstock zurück;
    Tanze;
else
    if Tanzende Biene in der Umgebung then
        Beobachte tanzende Biene;
        InformationFutterquelle = true;
    else
        Starte zufällige Suche nach der Futterquelle;
        if Suchlimit then
            Kehre zum Bienenstock zurück;
        else
            Finde die Futterquelle;
            InformationFutterquelle = true;
            Kehre zum Bienenstock zurück;
            Tanze;
        end
    end
end
```

**Algorithm 5:** Futtersuche Bienen

**Parameter Bienen** Im Modell ist die Anzahl der modellierten Bienen einstellbar.

## 4.2 Framework

Zur Implementierung der in Kapitel 4.1 definierten Modelle wird ein Framework verwendet. Für die ABM stehen mehrere Softwareplattformen und Bibliotheken zur Verfügung. Zu den bekanntesten und meistgenutzten Plattformen gehören NetLogo, MASON, Swarm und Repast.

### 4.2.1 Anforderungen

Die Anforderungen an das Framework zur Erstellung der in Kapitel 4.1 vorgestellten Modelle und anschließender grafischer Simulation sind:

- **Open-Source + kostenlos:** Der Quelltext des Frameworks ist öffentlich. Zudem kann das Framework kostenlos genutzt werden.
- **Aktualität:** Es ist eine aktuelle und stabile Version des Frameworks verfügbar.
- **Einsetzbarkeit:** Das Framework kann auf dem Betriebssystem Linux verwendet werden und unterstützt die Programmiersprache Java. Diese Anforderung entsteht durch die Präferenzen des Autors dieser Arbeit.
- **Visualisierung:** Modelle werden grafisch in 2D simuliert.
- **Dokumentation + Tutorial:** Das Framework ist umfangreich dokumentiert und besitzt ein Tutorial für Einsteiger.
- **Ausführungsgeschwindigkeit der Simulation:** Die Ausführungsgeschwindigkeit der Simulation ist hoch, trotz rechenintensiven Modellen mit vielen Agenten und Iterationen.

### 4.2.2 Übersicht

**NetLogo** NetLogo ist eine Open-Source-Plattform und wurde 1999 von U. Wilensky entwickelt. Die neuste Version 6.0.3 wurde im März 2018 veröffentlicht und steht für die Betriebssysteme Mac, Windows und Linux zur Verfügung. Die proprietäre funktionale Programmiersprache und die ausführliche Dokumentation machen es Einsteigern ohne Programmierkenntnisse einfach erste eigene Modelle zu implementieren. Das Fehlen objektorientierter Features schränkt die Funktionalität und Erweiterbarkeit des Frameworks jedoch ein. Darüber hinaus verfügt NetLogo über keine explizite Unterstützung für die Trennung zwischen Agenten, Agenten-Beziehungen und Agenten-Umgebung.

**MASON** Multi-Agent Simulator Of Networks (MASON) ist eine Java-Bibliothek, die ein breitgefächertes Repertoire an Funktionen für die ABM bereitstellt. Die aktuelle Version 1.9 wurde im Juni 2015 veröffentlicht. Eine umfangreiche Dokumentation der Open-Source-Bibliothek und ein einsteigerfreundliches Tutorial erlauben es innerhalb weniger Stunden eine erste lauffähige Simulation zu erstellen.

**Swarm** Swarm ist eine der ersten ABM-Plattformen. Ursprünglich wurde Swarm 1997 am Santa Fe Institute entwickelt und veröffentlicht. Die ABM-Plattform Swarm war zuerst nur auf UNIX-basierten Betriebssystemen einsetzbar und unterstützte nur die Programmiersprache Objective-C. Heute steht Swarm für die Betriebssysteme Mac, Windows und Linux zur Verfügung und unterstützt neben Objective-C auch Java. Das Framework ist Open-Source. Version 2.4.1 wurde im April 2009 veröffentlicht.

**Repast** Repast ist eine Open-Source-Plattform für die ABM. Die Software ist für die Betriebssysteme Mac, Windows und Linux verfügbar. Es existieren zwei verschiedene Editionen der Plattform. Repast Simphony implementiert grundlegende und erweiterte Konzepte der ABM. Unterstützt werden die Programmiersprachen Java, Logo und Groovy. Repast High Performance Computing ist ein agentenbasiertes Modellierungssystem speziell entwickelt für die parallele Ausführung der Simulationen auf verteilten Systemen. Zur Modellentwicklung mit dieser Edition von Repast werden umfangreiche Programmierkenntnisse in C++ vorausgesetzt. Die stabile Version 2.5 von Repast Symphonie wurde im Oktober 2017 veröffentlicht.

**Zusammenfassung** S. Railsback, S. Lytinen und S. Jackson[18] untersuchten verschiedene Plattformen für die ABM und Simulation. Ihre Ergebnisse über die oben genannten ABM-Plattformen können folgendermaßen zusammengefasst werden:

- **NetLogo:** Netlogo ist einfach zu bedienen und besitzt eine umfangreiche Dokumentation. NetLogo ist eine gute Wahl für simple Modelle bei denen die Ausführungszeit der Simulation eine untergeordnete Rolle spielt. Außerdem eignet sich NetLogo zur Erstellung eines Modellprototyps. Dieser Prototyp kann anschließend in anderen ABM-Plattformen erweitert werden.
- **MASON:** MASON ist eine gute Wahl für erfahrene Programmierer, die an rechenintensiven Modellen mit vielen Agenten und Iterationen arbeiten. Ein Vorteil von MASON ist die Reproduzierbarkeit der Simulationsergebnisse auf verschiedenen Betriebssystemen.
- **Swarm:** Swarm ist stabil, klein, gut organisiert und unterstützt komplexe Modelle. Voraussetzung für einen schnellen Einstieg ist Programmiererfahrung mit Objective-C.
- **Repast:** Repast ist die vollständigste ABM-Plattform und hat eine vergleichsweise schnelle Ausführungszeit der Simulationen. Die Dokumentation ist teilweise unvollständig.

[18] vgl. [8], S. Railsback, S. Lytinen, S. Jackson (2006)

### 4.2.3 Auswahl

Tabelle 1 zeigt eine Übersicht der Frameworks und die erfüllten Anforderungen. Zur Implementierung der in Kapitel 4.1 beschriebenen Modelle wird MASON als Framework gewählt. Mit MASON können die Modelle in Java implementiert werden. Da im Zuge dieser Studienarbeit rechenintensive Simulationen mit vielen Agenten durchgeführt werden sollen ist die hohe Ausführungsgeschwindigkeit der Simulationen wichtig. MASON bietet zudem eine umfangreiche Dokumentation und ein 14-stufiges Tutorial.

| | Open-Source + kostenlos | Aktualität | Einsetzbarkeit | Visualisierung | Doku + Tutorial | Geschwindigkeit |
|---|---|---|---|---|---|---|
| NetLogo | $\surd$ | Mrz. 2018 | $X$ | 2D/3D | $\surd$ | langsam |
| Mason | $\surd$ | Jun. 2015 | $\surd$ | 2D/3D | $\surd$ | sehr schnell |
| Swarm | $\surd$ | Apr. 2009 | $\surd$ | 2D/3D | $X$ | langsam |
| Repast | $\surd$ | Okt. 2017 | $\surd$ | 2D/3D | $\surd$ | schnell |

Tabelle 1: Auswahl des Frameworks

# 5 Implementierung

Im folgenden Kapitel wird die Implementierung der in Kapitel 4.1 definierten Modelle beschrieben. Auf eine Beschreibung der 2D-Visualisierung für die grafische Simulation der Modelle wird im Umfang dieser Arbeit verzichtet.

## 5.1 Finden des globalen Minimums der Rastrigin-Funktion mit der PSO

Die Implementierung des Modells basiert auf der Arbeit von A. Desai und S. Luke. Ihr Modell wurde unter der Academic Free License 3.0 veröffentlich. Das Modell darf modifiziert und anschließend veröffentlicht werden. Um die Implementierung des in Kapitel 4.1.1 definierten Modells zu erläutern wird die Implementierung anhand der Schritte des PSO-Algorithmus aus Kapitel 3.1.3 beschrieben.

### 5.1.1 Initalisiere Partikelschwarm im Suchraum

Im ersten Schritt des Modells wird die ausgewählte Anzahl von Partikeln an zufälligen Positionen im Suchraum mit zufälligen Ausgangsgeschwindigkeiten initialisiert. Listing 1 zeigt den Programmcode für die Initialisierung des Partikelschwarms.

```
public void start(){
  super.start();
  particles = new Particle[numParticles];
  space = new Continuous2D(height, width);
  //Waehle Rastrigin-Funktion
  Evaluatable f = mapFitnessFunction(fitnessFunction);
  //Initalisiere Partikel
  for (int i = 0; i < numParticles; i++){
    double x = (random.nextDouble() * width) - (width * 0.5);
    double y = (random.nextDouble() * height) - (height * 0.5);
    double vx = (random.nextDouble() * initialVelocityRange) - (
        initialVelocityRange * 0.5);
    double vy = (random.nextDouble() * initialVelocityRange) - (
        initialVelocityRange * 0.5);
    final Particle p = new Particle(x, y, vx, vy, this, f, i);
    particles[i] = p;
  }
}
```

Listing 1: PSO: Initalisierte Partikelschwarm im Suchraum

### 5.1.2 Bestimme Position und Geschwindigkeit

Nach der Initialisierung wird die Position und Geschwindigkeit der Partikel bestimmt. Listing 2 zeigt den Programmcode zur Bestimmung der Position und Geschwindigkeit.

```
public Particle(double x, double y, double vx, double vy, PSO pso, Evaluatable
    f) {
  super();
  //Bestimme Position und Geschwindigkeit der Partikel
  this.position.setTo(x, y);
  this.velocity.setTo(vx, vy);
  this.pso = pso;
  this.fitnessFunction = f;
  pso.space.setObjectLocation(this,new Double2D(position));
}
```

Listing 2: PSO: Bestimme Position und Geschwindigkeit

### 5.1.3 Bestimme den Fitnesswert

Das globale Minimum der Rastrigin-Funktion befindet sich bei den Koordinaten $x = 0$ mit $f(x) = 0$. Der Fitnesswert bestimmt die Güte der potentiellen Lösung eines Partikels. Von einem gewählten Maximalwert von 1000 wird durch die Berechnung in Listing 3 ein bestimmter Wert abgezogen. Dieser Wert ist abhängig von der Position des Partikels. Je näher die Position des Partikels zum globalen Minimum der Rastrigin-Funktion, desto größer ist der Fitnesswert des Partikels.

```
public class Rastrigin implements Evaluatable{
  private static final long serialVersionUID = 1;
  //Maximalwert 1000
  public double calcFitness(double x, double y){
    return (1000 - (20 + x*x - 10*Math.cos(2*Math.PI*x) + y*y - 10*Math.cos(2*
        Math.PI*y)));
  }
}
```

Listing 3: PSO: Bestimme Fitnesswert

### 5.1.4 Bestimme beste Position

Ist der aktuelle Fitnesswert eines Partikels größer als der in den vorausgehenden Schritten erreichte Fitnesswert des Partikels, dann handelt es sich um die bisher beste Position des Partikels. In jeder Iteration des Modells wird zudem der Partikel mit dem höchsten Fitnesswert bestimmt. Listing 4 prüft, ob die aktuelle Position die bisher beste Position des Partikels ist.

```
public void updateBest(double currVal, double currX, double currY){
  if (currVal > bestVal){
    bestVal = currVal;
    bestPosition.setTo(currX, currY);
    pso.updateBest(currVal, currX, currY);
  }
}
```

Listing 4: PSO: Bestimme beste Position

### 5.1.5 Berechne neue Geschwindigkeit

Listing 5 zeigt die getrennte Berechnung der x- und y-Komponente der neuen Geschwindigkeit des Partikels. Bei der Berechnung der neuen Geschwindigkeit werden die Vertrauenskoeffizienten berücksichtigt.

```
public void stepUpdateVelocity(){
  //Berechne x-Komponente
  double inertia = velocity.x;
  double pDelta = bestPosition.x - x;
  double gDelta = pso.bestPosition.x - x;
  double pWeight = pso.localTrust;
  double gWeight = pso.globalTrust;
  //Einfluss der Vertrauenskoeffizienten
  double vx = (0.9*inertia + pWeight*pDelta + gWeight*gDelta) / (1+pWeight+
      gWeight);
  velocity.setTo(vx, vy);
}
```

Listing 5: PSO: Berechne neue Geschwindigkeit

### 5.1.6 Berechne neue Position

Listing 6 zeigt den letzten Schritt einer Iteration des Modells. Für die nächste Iteration wird die neue Position des Partikels berechnet. Hierfür wird die in Listing 5 berechnete Geschwindigkeit mit der aktuellen Position des Partikels addiert.

```
public void stepUpdatePosition(){
  position.addIn(velocity);
  pso.space.setObjectLocation(this, new Double2D(position));
}
//Defintion von addIn
public final MutableDouble2D addIn(final MutableDouble2D other){
  x = other.x + x;
  y = other.y + y;
  return this;
}
```

Listing 6: PSO: Berechne neue Position

## 5.2 Die Futtersuche von Ameisen

Die Implementierung des Modells orientiert sich an der von Arbeit von L. Panait und S. Luke[19] und wird anhand der in Kapitel 4.1.2 definierten Methode beschrieben.

### 5.2.1 Kehre zum Ameisenbau zurück

Hat die modellierte Ameise die Futterquelle gefunden, dann kehrt sie zum Ameisenbau zurück. Dafür prüft der Programmcode in Listing 7 das Heimatpheromonniveau in ihrer Umgebung. Die Entscheidung für welchen Weg sich die modellierte Ameise entscheidet ist abhängig von den Parametern $\alpha$ und $\beta$.

```
if (hasFoodItem) {
  double max = -1; int max_x = x; int max_y = y;
  for(int dx = -1; dx < 2; dx++) {
    for(int dy = -1; dy < 2; dy++) {
      int _x = dx+x;
      int _y = dy+y;
      //Pruefe ob sich Position in der Umgebung befindet
      if ((dx == 0 && dy == 0) || _x < 0 || _y < 0 || _x >= AntsForage.
          GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT) continue;
      //Heimatpheromonkonzentration an der Position(_x,_y)
      double m = af.toHomeGrid.field[_x][_y];
      if (m > max || (m == max)
      {max = m; max_x = _x; max_y = _y;}
    }
  }
    //Siehe Kapitel 5.3.5 fuer Einfluss des Parameters beta
    //Siehe Kapitel 5.3.6 fuer Einfluss des Parameters alpha
    //Setze neue Position der modellierten Ameise
    af.buggrid.setObjectLocation(this, new Int2D(max_x, max_y));
    //Ameise befindet sich im Ameisenbau
    if (af.sites.field[max_x][max_y] == AntsForage.HOME)
    {hasFoodItem = ! hasFoodItem; }
}
```

Listing 7: Ameisen: Kehre zum Ameisenbau zurück

[19] http://cs.gmu.edu/ eclab/papers/panait04ant.pdf

### 5.2.2 Hinterlasse Futterpheromone

Auf der Rückkehr zum Ameisenbau hinterlässt die modellierte Ameise Futterpheromone, um den Weg zur Futterquelle zu markieren. Dafür wird das Futterpheromonniveau jeder besuchten Position, nach dem in Listing 8 abgebildeten Programmcode, erhöht.

```
if (hasFoodItem) {
  double max = af.toFoodGrid.field[x][y];
  int max_x = x;
  int max_y = y;
  for(int dx = -1; dx < 2; dx++){
    for(int dy = -1; dy < 2; dy++) {
      int _x = dx+x;
      int _y = dy+y;
      double m = af.toFoodGrid.field[_x][_y] * (dx * dy != 0 ? af.
          diagonalCutDown : af.updateCutDown);
      if (m > max) {max = m;}
    }
  }
  //Markiere Position mit Pheromonniveau
  af.toFoodGrid.field[x][y] = max;
}
```

Listing 8: Ameisen: Hinterlasse Futterpheromone

### 5.2.3 Finde Futterquelle

Listing 9 zeigt den Programmcode für die Orientierung der modellierten Ameise am Futterpheromonniveau der Positionen in ihrer direkten Umgebung. Der von der modellierten Ameise ausgewählte Weg ist abhängig von den Parametern $\alpha$ und $\beta$.

```
else {
  double max = -1;
  int max_x = x;
  int max_y = y;
  for(int dx = -1; dx < 2; dx++) {
    for(int dy = -1; dy < 2; dy++) {
      int _x = dx+x;
      int _y = dy+y;
      //Pruefe ob sich Position in der Umgebung befindet
      if ((dx == 0 && dy == 0) || _x < 0 || _y < 0 || _x >= AntsForage.
          GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT) continue;
      //Futterpheromonniveau an der Position(_x,_y)
      double m = af.toFoodGrid.field[_x][_y];
      if (m > max || (m == max && state.random.nextBoolean(1.0 / count++))){
        max = m; max_x = _x; max_y = _y;
      }
    }
  }
  //Siehe Kapitel 5.3.5 fuer Einfluss des Parameters beta
  //Siehe Kapitel 5.3.6 fuer Einfluss des Parameters alpha
  //Setze neue Position der modellierten Ameise
  af.buggrid.setObjectLocation(this, new Int2D(max_x, max_y));
  //Ameise befindet sich bei der Futterquelle
  if (af.sites.field[max_x][max_y] == AntsForage.FOOD)
  {hasFoodItem = ! hasFoodItem;}
}
```

Listing 9: Ameisen: Finde Futterquelle

### 5.2.4 Hinterlasse Heimatpheromone

Auf der Suche nach der Futterquelle hinterlässt die modellierte Ameise Heimatpheromone um den Weg zum Ameisenbau zu markieren. Dafür erhöht sie das Heimatpheromonniveau jeder besuchten Position, durch den in Listing 10 abgebildeten Programmcode.

```
else {
  double max = af.toHomeGrid.field[x][y];
  int max_x = x;
  int max_y = y;
  for(int dx = -1; dx < 2; dx++){
    for(int dy = -1; dy < 2; dy++){
      int _x = dx+x;
      int _y = dy+y;
      double m = af.toHomeGrid.field[_x][_y] * (dx * dy != 0 ? af.
          diagonalCutDown : af.updateCutDown);
      if (m > max) {max = m;}
    }
  }
  af.toHomeGrid.field[x][y] = max;
}
```

Listing 10: Ameisen: Hinterlasse Heimatpheromone

### 5.2.5 Parameter $\beta$

Parameter $\beta$ bestimmt mit welcher Wahrscheinlichkeit sich die modellierten Ameisen auf ihrem Weg an die Pheromonmarkierungen halten.

```
if (max == 0 && last != null) {
  if (state.random.nextBoolean(af.paramBeta)){
    int xm = x + (x - last.x);
    int ym = y + (y - last.y);
    if (xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.
        GRID_HEIGHT){
      max_x = xm; max_y = ym;
    }
  }
}
```

Listing 11: Ameisen: Parameter beta

### 5.2.6 Parameter $\alpha$

Parameter $\alpha$ bestimmt die Wahrscheinlichkeit mit der sich die modellierten Ameisen über die Pheromonmarkierungen hinwegsetzen und zufällig handeln.

```
else if (state.random.nextBoolean(af.paramAlpha)){
  int xd = (state.random.nextInt(3) - 1);
  int yd = (state.random.nextInt(3) - 1);
  int xm = x + xd; int ym = y + yd;
  if (!(xd == 0 && yd == 0) && xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0
       && ym < AntsForage.GRID_HEIGHT){
    max_x = xm; max_y = ym;
  }
}
```

Listing 12: Ameisen: Parameter alpha

### 5.2.7 Verdunstung der Pheromone

Zu jeder Iteration des Modells verdunsten die Pheromonmarkierungen auf der Agenten-Umgebung. Dazu wird ein konstanter Verdunstungsfaktor verwendet. Listing 13 zeigt die Programmcode zur Umsetzung der Verdunstung.

```
schedule.scheduleRepeating(Schedule.EPOCH,1, new Steppable(){
  //Verdunstung der Pheromonmarkierungen Verdunstungsfaktor pro Iteration
  public void step(SimState state) { toFoodGrid.multiply(evaporationConstant);
      toHomeGrid.multiply(evaporationConstant); }}, 1);
}
```

Listing 13: Ameisen: Verdunstung der Pheromone

## 5.3 Die Futtersuche von Bienen

Die Implementierung des Modells der Futtersuche von Bienen wird anhand der in Kapitel 4.1.2 definierten Methode beschrieben.

### 5.3.1 Zufällige Suche nach der Futterquelle

Befindet sich die modellierte Biene im Bienenstock und hat keine Information über die Futterquelle begibt sie sich mit einer niedrigen Wahrscheinlichkeit auf die zufallsgesteuerte Suche nach der Futterquelle. Die Suche ist begrenzt auf eine maximale Anzahl von Suchschritten. Findet die modellierte Biene die Futterquelle innerhalb der maximalen Anzahl an Suchschritten nicht, kehrt sie zum Bienenstock zurück. Listing 14 zeigt den Programmcode der zufallsgesteuerten Suche nach der Futterquelle.

```
protected void doStateInHiveWithoutInfo(){
  //Starte zufaellige Suche nach der Futterquelle mit geringer
      Wahrscheinlichkeit
  double pStartScouting = getSimulation().pStartScouting;
  if ((pStartScouting) >= r.nextDouble()){
    if (getFoodSource() == null){setState(State.scouting);}
  }
}

protected void doStateScouting(){
  if (r.nextInt(5) == 0) {
    turnBy(Math.toRadians(r.nextInt(40) - 20), Math.toRadians(r.nextInt(40) -
        20));
  }
  //Pruefung ob Futterquelle gefunden wurde
  foundSource();
  scoutSteps--;
  if (scoutSteps <= 0){setState(State.returnWithoutInfo);}
}
```

Listing 14: Bienen: Zufällige Suche nach der Futterquelle

### 5.3.2 Bewegung in Richtung der Futterquelle

Besitzt die modellierte Biene die Information über die Futterquelle fliegt sie zielgerichtet zu der ihr mitgeteilten, ungefähren Position der Futterquelle.

```
protected void doStateForaging(){
  sourceDirection.radius--;

  if (sourceDirection.radius <= 0){
    int maxSearchSteps = getSimulation().maxSearchSteps;
    searchSteps = maxSearchSteps;
    setState(State.searching);
  }
}
```

Listing 15: Bienen: Bewegung in Richtung der Futterquelle

### 5.3.3 Finden der Futterquelle

Befindet sich die modellierte Biene an der ihr mitgeteilten, ungefähren Position der Futterquelle. Startet sie die Suche nach der exakten Position der Futterquelle mittels dem in Listing 16 abgebildeten Programmcode. Findet die modellierte Biene die exakte Position nicht innerhalb der festgelegten Suchschritten, kehrt sie zum Bienenstock zurück.

```
protected void doStateSearching(){
  searchSteps--;
  if (searchSteps <= 0){setState(State.returnWithoutInfo);}
  turnBy(Math.toRadians(searchSteps - 2 * r.nextDouble() * searchSteps),
  Math.toRadians(searchSteps - 2 * r.nextDouble() * searchSteps));
  foundSource();
}
```

Listing 16: Bienen: Finden der Futterquelle

### 5.3.4 Rückkehr zum Bienenstock

Bricht die modellierte Biene ihre Suche ab oder hat die Futterquelle gefunden, dann kehrt sie nach dem in Listing 17 abgebildeten Programmcode zum Bienenstock zurück.

```
private void doStepReturning(Tuple3d targetLocation) {
  doStepFlying(targetLocation, true, false);
  sourceDirection.radius += getVelocityVector().length();
}

private void doStepFlying(Tuple3d targetLocation) {
  if (targetLocation != null) {
  headTo(targetLocation);
  }else {forward();}
}
```

Listing 17: Bienen:Rückkehr zum Bienenstock

### 5.3.5 Der Tanz

Um den modellierten Bienen, die sich im Bienenstock befinden und keine Information über die Futterquelle besitzen, die ungefähre Position der Futterquelle mitzuteilen tanzen die modellierten Bienen nach ihrer Rückkehr von der Futterquelle.

```
protected void doStateDancing() {
  dancingTime--;
  if (dancingTime <= 0) {
    setState(State.foraging);
  }
}
```

Listing 18: Bienen: Der Tanz

### 5.3.6 Beobachtung des Tanzes

Die modellierten Bienen, die sich im Bienenstock befinden und keine Information über die Futterquelle besitzen, beobachten den Tanz um die Information über die Futterquelle zu erhalten. Nachdem die modellierte Biene einen Tanz beobachtet hat besitzt sie die Information über die Futterquelle und fliegt zur ungefähren Position der Futterquelle. Listing 19 zeigt den Programmcode mit dem sich die beobachtende Biene die Informationen über die Futterquelle kopiert.

```
private void listenToDancingBee(){
  IMovingAgent[] agents = this.getObjectsWithinMyDistance(1.0, true,
  true, this.getSphereRadius(), false, null);
  agents = (IMovingAgent[]) Filter.filter(agents, Bee.class);
  if (agents.length > 0){
    int index = r.nextInt(agents.length);
    Bee b = (Bee) agents[index];
    if (b.getState() == Bee.State.dancing){
      copySourceInformationFrom(b);
      sourceDirection.radius += Math.round(r.nextGaussian() * sourceDirection.
          radius);
      double angle;
      angle = sourceDirection.azimuth;
      angle += (r.nextGaussian() * Math.PI;
      Geometric.clampAngleRadians(angle);
      sourceDirection.azimuth = angle;
      setTargetLocation(sourceDirection);
      setState(State.foraging);
    }
  }
}
```

Listing 19: Bienen: Beobachtung des Tanzes

# 6 Experimente

Im folgenden Kapitel werden die Ergebnisse, der im Rahmen dieser Arbeit durchgeführten Simulationen, dokumentiert. Dafür werden die Simulationen der in Kapitel 4.1 beschriebenen Modelle mit verschiedenen Parametern ausgeführt.

## 6.1 Finden des globalen Minimums der Rastrigin-Funktion mit PSO

Das Finden des globalen Minimums der Rastrigin-Funktion stellt aufgrund ihres großen Suchraums und der hohen Anzahl lokaler Minima ein schweres Optimierungsproblem dar. Die in Kapitel 5.1 beschriebene Implementierung der PSO eignet erzielt in der Simulation sehr gute näherungsweise Lösungen. Die in den Simulationen erzielten Ergebnisse befinden sich im Anhang A.1.

### 6.1.1 Auswirkungen der Parameter auf die Ergebnisse der Simulation

In der Simulation zeigt sich, dass die Güte der erzielten Lösungen abhängig von der Anzahl an Partikeln ist. Umso höher die Anzahl an Partikel, desto bessere Lösungen werden erzielt. Die erzielten Ergebnisse in der Simulationen zeigen zudem, dass die Güte der Lösungen mit einem hohen lokalen Vertrauenskoeffizienten zunimmt. Bei einem niedrigen lokalen Vertrauenskoeffizienten und hohen globalen Vertrauenskoeffizienten werden insgesamt schlechtere Lösungen konstruiert.

## 6.2 Futtersuche von Ameisen

Im Modell der Futtersuche von Ameisen bilden die modellierten Ameisen, wie auch echte Ameisen, eine Ameisenstraße vom Ameisenbau zur Futterquelle. Die sich, in der Simulation, bildende Ameisenstraße ist in Abbildung 5 dargestellt. Dieser Effekt tritt auf, da die modellierten Ameisen den Pheromonmarkierungen folgen. Die Ergebnisse der durchgeführten Simulationen befinden sich in Anhang A.2.

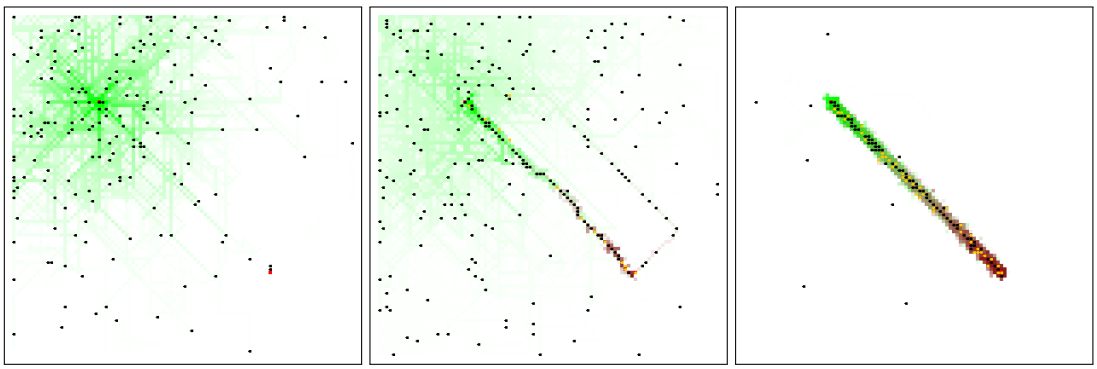

Abbildung 5: Bildung einer Ameisenstraße in der Simulation der Futtersuche von Ameisen

### 6.2.1 Auswirkungen der Parameter auf die Ergebnisse der Simulation

Die Strategie der Ameisen eine direkte, durch Pheromone markierte, Verbindung zwischen Ameisenbau und Futterquelle zu bilden wird durch den Parameter $\alpha$ gestört. Abbildung 6 zeigt den Einfluss des Parameters $\alpha$. Die zufallsgesteuerte Suche nach der Futterquelle ist zur Beginn der

Simulation jedoch notwendig, da keine Pheromonmarkierungen vorhanden sind. Wird die Simulation der Futtersuche mit einer geringen Anzahl an modellierten Ameisen durchgeführt, dauert es länger bis die Futterquelle von einer Ameise gefunden wird. Bei einer hohen Anzahl an modellierten Ameisen steigt die Wahrscheinlichkeit, dass die modellierten Ameisen die Futterquelle finden. In der Simulation werden die meisten Futtereinheiten mit geringem Parameter $\alpha$ gesammelt.

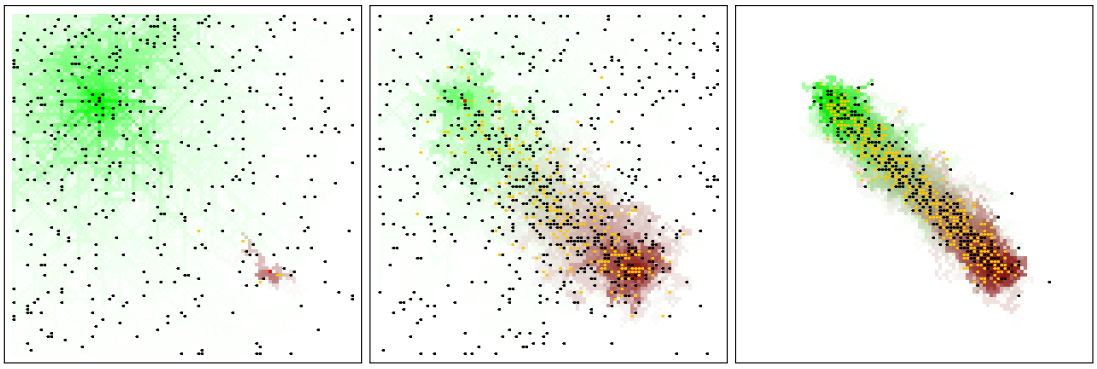

Abbildung 6: Einfluss des Parameters $\alpha$ in der Simulation der Futtersuche von Ameisen

## 6.3 Futtersuche von Bienen

Wie in der Natur tanzen die modellierten Bienen, um anderen Bienen im Bienenstock die Richtung und Entfernung der Futterquelle zu zeigen. Da die Bienen im Bienenstock lediglich tanzende Bienen in ihrer direkten Umgebung sehen können, veranlasst eine tanzende Biene nur eine geringe Anzahl an Bienen dazu, ihr zur Futterquelle zu folgen. Kehren die rekrutierten Bienen erfolgreich von der Futterquelle zurück tanzen auch sie. Es entsteht eine Art Schneeballeffekt. Je mehr Bienen tanzen desto mehr Bienen bekommen die Information über die Richtung und Entfernung zur Futterquelle. Abbildung 7 zeigt den Abbau der Futterquelle in der Simulation.

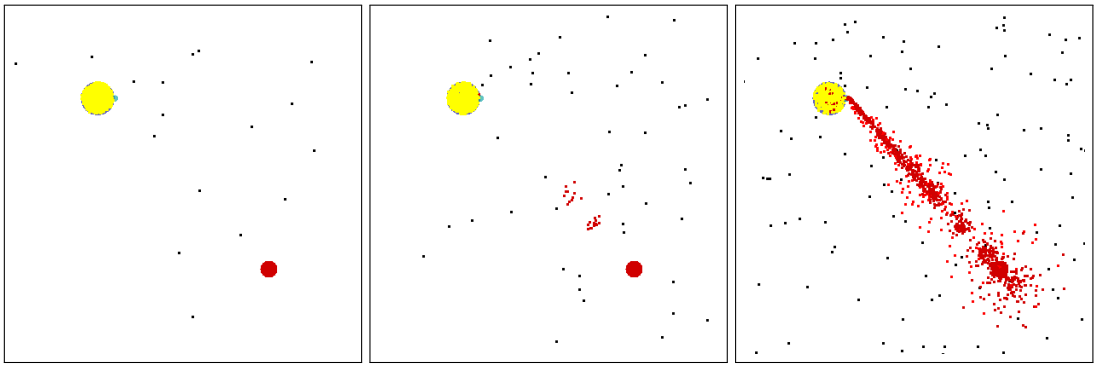

Abbildung 7: Abbau der Futterquelle in der Simulation der Futtersuche von Bienen

# 7 Zusammenfassung

Die Futtersuche von Ameisen und Bienen, anhand eines agentenbasierten Modells zu simulieren ist gelungen. Die Simulationen zeigen die typischen Strategien der Ameisen- und Bienenkolonien zum effizienten Abbau einer Futterquelle. Während die einzelnen modellierten Agenten nur über ein sehr beschränktes Repertoire an Fähigkeiten und Entscheidungsmöglichkeiten verfügen, entstehen bei einem Kollektiv an modellierten Agenten intelligente Lösungen auf Makroebene. Die modellierten Agenten besitzen dabei nur eine eingeschränkte Sicht auf das Gesamtsystem. Sie verfügen nur über die Informationen in ihrer unmittelbaren Umgebung. Wie in der Natur handeln die modellierten Agenten autonom. Sie entscheiden situationsbedingt welche der ihnen zur Verfügung stehenden Aktionen auszuführen ist.

Das in dieser Arbeit implementierte Modell der PSO zum Finden des globalen Minimums der Rastrigin-Funktion liefert gute näherungsweise Lösungen. Das beste Resultat der im Rahmen dieser Arbeit durchgeführten Simulationen ist die näherungsweise Lösung $x = 0.0008121$ mit $f(x) = 0.00052893$, während sich das globale Minimum der Rastrigin-Funktion bei $x = 0$ mit $f(x) = 0$ befindet.

# Literatur


[1] A. Aulinger. Kollektive Intelligenz: Methoden, Erfahrungen und Perspektiven. Steinbeis-Edition. ISBN: 978-3941317151. Entstehungsjahr: 2009.

[2] E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. National Academy of Sciences. 99 (suppl 3) 7280-7287. Entstehungsjahr: 2002.

[3] M. Dorigo, M. Birattari, T. Stützle. Ant Coloncy Optimization. Computational Intelligence Magazine, pp 28-39. Entstehungsjahr: 2004.

[4] J. Kennedy, R. Eberhart. A New Optimizer Using Particle Swarm Theory. MHS'95, Proceedings of the Sixth International Symposium. Entstehungsjahr: 1995.

[5] Y. Liu. Partikelschwarmoptimierung für diskrete Probleme. Technische Universität München. Entstehungsjahr: 2014.

Weblink: http://%3A%2F%2Fwwwmayr.informatik.tu-muenchen.de%2Fkonferenzen%2FFerienakademie14%2Fslides_papers%2Fpaper_Yushan_Liu.pdf&usg=AOvVaw0y3oo34w9QhrreOniL4ILk
Einsichtnahme: 14.01.2018

[6] C. Macal, M. North. Tutorial on agent-based modelling and simulation. Journal of Simulation 4, Nr. 1, pp 151-162. Entstehungsjahr: 2010.

[7] L. Pintscher. Schwarmintelligenz. Universität Karlsruhe. Entstehungsjahr: ohne Jahr.

Weblink: http://lydiapintscher.de/uni/schwarmintelligenz.pdf
Einsichtnahme: 14.01.2018

[8] S. Railsback, S. Lytinen, S. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations. SIMULATION. Vol 82, Issue 9. Entstehungsjahr: 2006.

[9] S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River. Entstehungsjahr: 2009.

[10] D. Teodorović, P. Lučić. Bee system: Modeling combinatorial optimization transportation engineering problems by swarm intelligence. Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis, pp 441-445. Entstehungsjahr: 2001.

[11] R. Wehner. Miniaturgehirne und kollektive Intelligenz. Zur Evolution biologischer Komplexität. Züricher Universitätszeitschriften 3. Entstehungsjahr: 2001.

[12] W.M. Wheeler. The Ant Colony as an Organism. Journal of Morphology Volume 22, Issue 2, Seite 307-325. Entstehungsjahr: 1911.

Weblink: http://www3.interscience.wiley.com/journal/109914213/abstract
Einsichtnahme: 04.01.2018

[13] Wikipedia. Rastrigin-Funktion. Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 1. Sep 2016, 07:34 UTC.

Weblink:https://de.wikipedia.org/w/index.php?title=Rastrigin-Funktion&oldid=177349710
Einsichtnahme: 02.03.2018

[14] M. Wooldridge, N. Jennings. Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10, Nr. 2, pp 115-152. Entstehungsjahr: 1995.

# Verzeichnis der Anhänge

**A Anhang** **30**
A.1 Experimente PSO . . . . . 30
A.1.1 Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7 30
A.1.2 Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5 31
A.1.3 Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3 32
A.2 Experimente Futtersuche von Ameisen . . . . . 33
A.2.1 $\alpha$ = 0.3, $\beta$ = 0.7 . . . . . 33
A.2.2 $\alpha$ = 0.5, $\beta$ = 0.5 . . . . . 34
A.2.3 $\alpha$ = 0.7, $\beta$ = 0.3 . . . . . 35
A.3 Experimente Futtersuche von Bienen . . . . . 36

# A Anhang

## A.1 Experimente PSO

### A.1.1 Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7

- 1000 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 0.01262632028506161,-0.013384001669769718 |
| 2 | 0.0019848665525268316,0.000528938028764081 |
| 3 | 0.0008120558872288797,0.00047645837213217135 |
| 4 | 0.001151650858115083,0.002847968150353708 |

Tabelle 2: Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7

- 100 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 0.9772663170332931,-0.015964212524956722 |
| 2 | -0.012299255721850477,-0.033490171619710574 |
| 3 | -0.012308005422580148,0.9809504459943033 |
| 4 | -0.050978886961883685,0.9850664961084877 |

Tabelle 3: Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.3, lokaler Vertrauenskoeffizient = 0.7

### A.1.2 Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5

- 1000 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 0.01262632028506161,-0.013384001669769718 |
| 2 | 0.0019848665525268316,0.000528938028764081 |
| 3 | 0.0008120558872288797,0.00047645837213217135 |
| 4 | 0.001151650858115083,0.002847968150353708 |

Tabelle 4: Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5

- 100 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | -0.020958980636341273,-0.01571132254960461 |
| 2 | -0.032716891537396986,-0.008106273053844149 |
| 3 | -0.01916372314789072,-0.9962361142871572 |
| 4 | 0.005898689367579202,-0.025737966396528478 |

Tabelle 5: Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.5, lokaler Vertrauenskoeffizient = 0.5

### A.1.3 Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3

- 1000 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 0.01262632028506161,-0.013384001669769718 |
| 2 | 0.0019848665525268316,0.000528938028764081 |
| 3 | 0.0008120558872288797,0.00047645837213217135 |
| 4 | 0.001151650858115083,0.002847968150353708 |

Tabelle 6: Ergebnisse Simulation PSO: 1000 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3

- 100 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 0.9772663170332931,-0.015964212524956722 |
| 2 | -0.012299255721850477,-0.033490171619710574 |
| 3 | -0.012308005422580148,0.9809504459943033 |
| 4 | -0.050978886961883685,0.9850664961084877 |

Tabelle 7: Ergebnisse Simulation PSO: 100 Partikel, 100 Iterationen, Globaler Vertrauenskoeffizient = 0.7, lokaler Vertrauenskoeffizient = 0.3

## A.2 Experimente Futtersuche von Ameisen

### A.2.1 $\alpha$ = 0.3, $\beta$ = 0.7

- 1000 Ameisen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 27190 |
| 2 | 25918 |
| 3 | 25535 |
| 4 | 25841 |

Tabelle 8: Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.3, $\beta$ = 0.7

- 100 Ameisen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 2453 |
| 2 | 2465 |
| 3 | 2528 |
| 4 | 2529 |

Tabelle 9: Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.3, $\beta$ = 0.7

### A.2.2 $\alpha$ = 0.5, $\beta$ = 0.5

- 1000 Ameisen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 16106 |
| 2 | 16758 |
| 3 | 16504 |
| 4 | 15972 |

Tabelle 10: Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.5, $\beta$ = 0.5

- 100 Ameisen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 1110 |
| 2 | 1428 |
| 3 | 996 |
| 4 | 1277 |

Tabelle 11: Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.5, $\beta$ = 0.5

### A.2.3 $\alpha$ = 0.7, $\beta$ = 0.3

- 1000 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 9951 |
| 2 | 9628 |
| 3 | 9428 |
| 4 | 8923 |

Tabelle 12: Ergebnisse Simulation der Futtersuche von Ameisen: 1000 Ameisen, 5000 Iterationen, $\alpha$ = 0.7, $\beta$ = 0.3

- 100 Partikel, 100 Iterationen

| Simulation | beste Lösung (Position(x,y)) |
|---|---|
| 1 | 709 |
| 2 | 595 |
| 3 | 812 |
| 4 | 772 |

Tabelle 13: Ergebnisse Simulation der Futtersuche von Ameisen: 100 Ameisen, 5000 Iterationen, $\alpha$ = 0.7, $\beta$ = 0.3

## A.3 Experimente Futtersuche von Bienen

- 2000 Bienen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 1822 |
| 2 | 2152 |
| 3 | 2395 |
| 4 | 1841 |

Tabelle 14: Ergebnisse Simulation der Futtersuche von Bienen: 2000 Bienen, 5000 Iterationen

- 1000 Bienen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 805 |
| 2 | 1146 |
| 3 | 830 |
| 4 | 996 |

Tabelle 15: Ergebnisse Simulation der Futtersuche von Bienen: 1000 Bienen, 5000 Iterationen

- 100 Ameisen, 5000 Iterationen

| Simulation | gesammelte Futtereinheiten |
|---|---|
| 1 | 11 |
| 2 | 4 |
| 3 | 40 |
| 4 | 69 |

Tabelle 16: Ergebnisse Simulation der Futtersuche von Bienen: 100 Bienen, 5000 Iterationen